Thesis for The Degree of Licentiate of Engineering

Techniques to Save Energy in Heterogeneous Multicore Architectures under QoS Constraints

Muhammad Waqar Azhar

Division of Computer Engineering
Department of Computer Science & Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden, 2019

Techniques to Save Energy in Heterogeneous Multicore Architectures under QoS Constraints

Muhammad Waqar Azhar

Technical Report No 193L
ISSN 1652-876X
Department of Computer Science & Engineering
Division of Computer Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2019.

"The more you know, the more you know you don't know."
- Aristotle

Abstract

Typically, applications are run with available system resources leading to over-provisioning of resources which can lead to high energy consumption. If the computational demand is specified, in terms of a Quality of Service (QoS) contract, it is possible to devote just enough resources to applications and thereby reduce energy consumption. Modern heterogeneous multicore platforms, such as ARM big.LITTLE, typically provide a multidimensional space of resources, called configuration space, such as Voltage-Frequency (V-F) settings, thread count and processor types, which can be configured at run-time to open up new opportunities for resource management.

This thesis presents techniques to improve energy efficiency under the constraint of QoS by managing the resource allocation at run-time for applications run on heterogeneous multicore platforms. The applications considered are iterative with a computational deadline associated with each loop iteration. The proposed techniques apply to a framework that uses applications' outer loop iterations as a means for progress-tracking and prediction of the execution time.

A first contribution of the thesis is a resource management technique for single-threaded applications that uses core type (e.g. big or little cores) and V-F settings as a configuration space to select a configuration, for each iteration, based on the execution-time prediction of future iterations and computational deadlines. The thesis shows that an energy saving of 25% over the *race-to-idle* state-of-the-art technique is achieved without missing any deadlines. This scheme incurs only 0.6% and 0.8% of timing and energy overheads, respectively.

A second contribution of the thesis is a novel resource-management policy for multi-threaded applications. Here, the configuration space is extended to also consider the thread count, i.e., the number of cores assigned to multi-threaded applications. The proposed technique first chooses the most energy-efficient configuration that meets the computational deadline. Since an iteration typically finishes before the deadline, the proposed technique collects the generated execution-time slack over subsequent iterations with the goal of selecting a configuration that can save more energy. To allow for on-line exploration of the configuration space, at low overhead, a third contribution of the thesis is an online, low-overhead prediction method based on interpolation, that measures the execution statistics at end points of each configuration-space dimension and interpolates the values at intermediate configurations. Overall, the proposed technique saves 61% energy compared to the state-of-the-art *race-to-idle* technique without missing any deadlines. Further, it only incurs 0.6% and 0.7% of timing and energy overheads, respectively.

Keywords:

Energy Efficiency, Quality-of-Service (QoS), Resource Management, Heterogeneous Multicores, DVFS

Acknowledgment

I would first like to thank my supervisor Professor Per Stenström, who provided me with extensive professional guidance and taught me a great deal about scientific research. I will continue to benefit from his insightful comments and encouragement for the rest of my Ph.D. studies.

I am also grateful to my previous co-supervisor Vassilis Papaefstathiou, whose input and feedback steered me in the right direction during the initial part of my Ph.D.

I am also indebted to my other co-supervisor Miquel Pericas whose thoughtful and precise feedback enabled me to polish my research ideas further.

I would particularly like to thank Professor Per Larsson-Edefors with whom I began my journey as a researcher during my years of Masters studies, and who incented me to widen my research from various perspectives.

I would like to acknowledge my colleagues, Petros, Ahsen, Prajith, Pedro, Stavros, Evangelos, Albin, Stefano, Madhavan, Ioannis, Jan, Monica, and others. A special thanks to Mehrzad with whom I had countless stimulating discussions regarding novel ideas.

Finally, I must express my very profound gratitude to my parents and to my wife for providing me with unwavering support and continuous encouragement throughout my years of study and through the process of writing this thesis. This accomplishment would not have been possible without them. Thank you.

This work would not have been possible without the research grants from the Swedish Research Council and the European Research Council through the MECCA project.

List of Publications

Appended publications

This thesis is based on the following publications:

I M. Waqar Azhar, Per Stenström, and Vassilis Papaefstathiou "SLOOP: QoS-Supervised Loop Execution to Reduce Energy on Heterogeneous Architectures"
ACM Transactions on Architecture and Code Optimization (TACO), Article 41 Volume-14, Issue-4, December 2017.

II M Waqar Azhar, Miquel Pericas, Per Stenström "SaC: Exploiting Execution-Time Slack to Save Energy in Heterogeneous Multicore Systems"
Technical report 2019:02, ISSN 1652-926X
In submission to The 48th International Conference on Parallel Processing (ICPP 2019).

The papers will be referred to in the thesis using their Roman numerals.

Contents

Chapter 1

Introduction

Energy efficiency in computing systems is of paramount importance especially with the end of Dennard scaling [1, 2]. Energy efficiency is a sought-after goal across the entire computing spectrum. However, it is of particular relevance to battery-powered systems, e.g., smartphones, and data centers, that consume significant portions of electrical power. Energy efficiency can lead to longer operating times in mobile devices while data-center infrastructures can reap cost benefits associated with lower power consumption and removal of excess heat [3, 4].

Performance improvements for single-threaded applications experienced a slowdown due to high energy consumption in the beginning of this millenium which led to the shift to the multicore era. In multicores, multiple computational resources (or processors) are available in a single chip [5]. Computing applications usually have a variety of characteristics, each requiring a specific microarchitecture to meet the desired performance and energy-efficiency goals. Therefore, integrating various types of processors into a single chip, known as heterogeneous multicore architectures, is beneficial, where the heterogeneity comes in multiple forms. On the one hand, there exist processors with different performance levels packaged as, e.g., in-order or out-of-order processors, but capable of executing the same Instruction Set Architecture (ISA). On the other hand, processors with different ISAs, e.g., graphical processing units (GPUs), and special-purpose computational resources, called accelerators, are typically added to enrich heterogeneity [6, 7]. This rich diversity is suitable to meet performance and energy efficiency goals provided that a resource management system can allocate appropriate computational resources to applications.

Traditionally, applications execute without any notion of a computational deadline or Quality of service (QoS) typically using available computational resources. This *best-effort approach* wastes energy as certain types of applications need only produce results at a certain rate, or at specific deadlines, and computing faster than these deadlines do not add any value to the user. These deadlines are typically governed by real-world requirements, e.g. a certain video frame rate. Allocating a sufficient amount of computational resources to an application so that it can finish *just-in-time* before the deadline, can potentially improve energy efficiency. To accomplish this, a resource management framework is needed that is capable of predicting the application behaviour at run-time and uses the QoS specification to select an appropriate resource allocation.

This thesis proposes techniques for on-line resource management on heterogeneous multiprocessing platforms under QoS constraints to improve energy efficiency. The methods are centered around the hypothesis that allocating a sufficient amount of computational resources to meet QoS constraints can improve energy efficiency substantially. The approach taken is to monitor application behavior at run-time and use the QoS specification given by the programmer to allow the resource manager (RM) to predict what resources are needed by the application.

This thesis is based on two papers. **Paper I** proposes a framework for monitoring, predicting and scheduling iterative sequential applications on a heterogeneous platform, where applications are allocated an appropriate amount of resources in terms of voltage-frequency (V-F) and processor types so that they finish on the deadline and thus save energy. **Paper II** extends this framework to multi-threaded iterative applications executing on a

heterogeneous platform, where the resource manager controls the number of processors, processor types and voltage-frequency to improve energy efficiency.

Chapter 2

Summary of the Papers

2.1 Summary of Paper I

Power gating (or *race-to-idle*) is the predominantly used method to save energy, where the resource manager first executes the application using the available resources and then switches to a power-down state (or idle) [8, 9]. Application execution may consist of phases of alternating periods of computation and periods of inactivity. Inactivity may occur between consecutive computational phases, or at the end of execution. However, in both cases, a substantial amount of energy is consumed during computation as available resources are used in that period. *Race-to-idle* techniques are useful in scenarios where the idle periods are significant. In fact, it has been proposed to lengthen the idle period by *sprinting* to completion by momentarily exceeding the power budget [10]. Again, the extra energy consumed during the *sprinting* phase is, in general, difficult to be replenished by idling.

Applications also show a varying behaviour inside the computational phases where variability in factors, e.g., instruction level parallelism (ILP) and memory level parallelism (MLP) change the computational demand. Hence, instead of powering down the complete processor, some approaches propose to allocate the computational resources based on the computational demand [11–13].

In the context of *Paper I*, today's heterogeneous multicore architectures [6, 14] provide the opportunity to allocate resources, such as processor types and the frequency at which they run (Dynamic Voltage Frequency Scaling – DVFS), to applications. The schemes that adjust the resource allocation to the computational demand typically offer the same performance as the *race-to-idle* technique and use the application behavioral prediction to minimize the stall cycles by either adjusting DVFS [11, 12] or allocate it to specific processor types [13]. Since only a limited amount of such variations, e.g., ILP and MLP, are available, energy savings are also limited.

Introducing QoS unlocks further opportunities for energy saving, where the resource manager can allocate the appropriate amount of resources to applications in order to meet their computational deadlines. Hughes et al. [15] propose to assign a specific voltage-frequency setting and architectural configuration to meet a certain video frame-rate. Similarly, Kluge et al. [16] suggest using DVFS to regulate the computational speed in video encoding/decoding applications. These techniques are based on specific application insights and are not application agnostic. Sue et al. [17] propose to regulate instructions per second (IPS) to a predefined level based on average or worst case execution, in order to improve energy efficiency. Assigning the same computational demand to different computational phases often leads to inadequate allocation of resources.

Energy efficiency can be improved by predicting the behavior of application phases at run time and adjusting the computational resources allocated to each of them, so that each phase completes just in time before the deadline. This requires a framework for progress-tracking and prediction of the application behavior and for allocating a sufficient amount of resources to meet the QoS requirement.

Paper I proposes a framework, called SLOOP, for resource management under QoS constraints for iterative sequential applications executing on heterogeneous multicores by controlling DVFS and processor type. An outer loop typically encapsulates the behavior of

the program. Each iteration computes a new output by applying the application kernel on the new input data, and the QoS specification enforces an upper limit on the completion time. **Paper I** proposes to use the loop iteration boundaries as execution points to monitor and predict the application behaviour. A novel scheduler uses the prediction, along with the deadline, to allocate an appropriate amount of architectural resources to subsequent iterations.

Instruction and cycle counts from hardware performance counters are used to monitor the application behavior at iteration boundaries. This information is used to predict the execution time in future iterations. The instruction count from previous iterations is first used to predict the instruction count for the next iteration using two types of predictors: an average predictor and a gradient predictor. These instruction-count predictors are used to predict the execution time at any given voltage-frequency (V-F) setting and on any type of processor. The cycle count is used for progress tracking and at the end of each iteration, the execution time slack (or simply *slack*) is calculated as the difference between execution time and deadline. By keeping track of the accumulated slack, the deadlines for the subsequent iterations can be extended accordingly, and this can be used to reduce the amount of resources further to save more energy. The scheduler can use the execution time predictions, slack and the QoS specification to compute a suitable configuration in terms of processor type and V-F to execute subsequent loop iterations.

Various processor types in heterogeneous systems have different performance levels and can fulfill the given QoS requirement at different V-F settings, with lower V-F states offering more energy efficiency. The scheduling algorithm uses this insight to predict the minimum V-F setting that can meet the deadline for each processor type and then uses the most energy-efficient processor type and V-F setting. This process of prediction and resource allocation repeats after each iteration. Eventually, each iteration finishes before the deadline. In a nutshell, the resource allocation is adjusted to the dynamic computational demand.

The SLOOP framework is evaluated on an ODROID-XU3 board containing Samsung Exynos-5422 chipset [18] that is based on ARM's big.LITTLE technology using applications from the ALPBench [19] and SPEC2006 [20] benchmark suites. The deadline is defined using the *race-to-idle* technique and it is set equal to the execution time of the slowest iteration among all the iterations of application's main loop. This deadline setting makes sure that all the iterations meet the deadline while executing at highest resource allocation.

The SLOOP scheme is evaluated against race-to-idle as the baseline, the scheme proposed in state-of-the-art by Sue et al. [17] (henceforth referred to as Sue) and an oracle scheme, denoted *Oracle*. Here, *Oracle* refers to a scheme that makes a perfect prediction and uses the SLOOP's scheduler. Sue's scheme has two variants: the average profile and worst-case profile, where the former uses the average instruction count and the latter uses the worst-case instruction count as prediction and these values are computed using the instruction count per iteration trace of baseline. Both schemes use the SLOOP's scheduler.

The proposed *SLOOP* scheme saves 25% energy compared to the *race-to-idle* and is only 8% worse off than *Oracle*, without missing any deadlines. Sue's worst-case profile method performs similarly with the *race-to-idle* and does not save any energy while the Sue's average profile method saves 24 % energy compared to the baseline but misses 97% of the deadlines. Moreover, the SLOOP scheme only incurs 0.06% and 0.07% of timing and energy overheads, respectively. By further relaxing the deadline by 20%, 50% and 100% the energy savings increase to 42%, 52% and 63% respectively.

2.2 Summary of Paper II

Single-threaded performance improvements have saturated over the last decade and this has led us to the multicore era where performance improvements are harnessed by thread-level parallelism (TLP) using many processors, often of different types, on a single chip. Multi-threaded programs executing on heterogeneous multicores add another resource, i.e. thread count, to the existing resources such as DVFS and type of processors. However, parallel programming models that were developed to exploit TLP typically leave resource management to the run-time system or the programmer.

Race-to-idle over-provisions the resource allocation. On the other hand, QoS requirements enable allocation of a sufficient amount of resources to meet the computational deadlines with less energy. In this context, several techniques [21–23] propose to slow down the execution to finish just before the deadline resulting in execution time slack. Computational deadlines only allow a subset of the configuration space to be scheduled. Unfortunately, configurations

in a heterogeneous multicore system that do not meet the deadline are often more energy efficient, for example using lower frequencies or in-order cores. The execution-time slack can be used to expose these energy efficient configurations and further increase the energy efficiency.

Whether a scheme uses slack [22] or not [21, 23], an on-line prediction method to estimate the performance and energy behavior of the application with respect to resource allocation is necessary. Earlier work has used machine learning (ML) [21, 23], where these ML-based approaches require a considerable amount of training. Furthermore, off-line training-based approaches [23] yield high error rates when exposed to unknown scenarios, whereas in case of on-line training [21] the overheads at the level of 10% become a concern. In contrast, Li et al. [22] provide a heuristic-based online method but only consider a small portion of the configuration space that was pruned earlier during the training phase.

The full potential of energy savings can only be exploited by considering the entire configuration space for scheduling. Moreover, to enable this, a lightweight online prediction method must be used to curtail the overheads.

Paper II proposes a resource management policy based on *execution-time slack*, referred to as *SaC* - Slack as a Currency, where the RM first allocates a sufficient amount of resources to an application so that it meets the deadline. Second, as slack builds up, the RM uses slack to allocate more energy-efficient configurations to the application that would normally not meet the deadline. To enable the resource allocation, this paper further proposes a lightweight online prediction method. Similar to **Paper I**, this paper also uses the iterations of the applications' outer loop to monitor the application behavior and schedule it on a new configuration.

The prediction method is based on monitoring application behaviour (instructions per second (IPS) and energy per instruction (EPI)) at pre-defined configurations and uses interpolation to estimate the behaviour at the remaining configurations. For example, the application is executed using a maximum number of threads at maximum and minimum frequencies in two consecutive iterations, thereby measuring IPS and EPI values. The IPS and EPI, at frequencies between the minimum and maximum frequency, is computed using interpolation.

During training, the first two iterations are executed using maximum thread count at the maximum and minimum frequencies respectively. The third and fourth iterations then execute using the minimum thread count at maximum and minimum frequencies respectively. Since each processor type in the system has a specific performance and energy characteristic, the same training procedure is repeated for each processor type.

During steady state, the resource manager measures execution statistics, including instruction count, cycle count and energy consumption, at the end of every iteration and hence finds a suitable configuration to execute the next iteration. A history of the instruction count from previous iterations is maintained and used for predicting the instruction count for future iterations, referred to as workload prediction. The cycle count is used to compute the execution time and slack. The resource manager keeps track of the accumulated slack.

The resource manager uses the workload prediction, accumulated slack and deadlines to select a configuration to execute the subsequent iteration. First, an energy-efficient configuration, called slack generating configuration (SGC), is selected that meets the deadline without considering slack. Then, slack is allowed to build up across multiple iterations and a new configuration, called the slack using configuration (SUC), is selected by relaxing the deadline by the accumulated slack. Since it is desirable to execute multiple iterations in slack using (SU) mode, the accumulated slack is distributed over several iterations. Once the slack reaches close to zero, a new SGC is selected and this process repeats over and over again.

The method of selecting the SGC and SUC is based on the fact that, for a specific thread count and processor type, the minimum voltage-frequency that meets the deadline yields minimum energy. So the minimum frequency for all the combinations of processor types and thread counts is computed. The configuration with the minimum predicted energy is picked.

To select a SUC, the same analysis is repeated but by relaxing the deadline by the amount of accumulated slack. However, the method of predicting the energy is different in case of SUC. Different SUCs consume the accumulated slack in different number of iterations. Thus, a configuration with lower EPI, but a smaller number of iterations in SU mode, will have a lesser impact on energy efficiency compared to a configuration with slightly higher EPI and more iterations in SU mode. In short, the iteration count for all the SUCs is predicted leading to the prediction of average EPI. Finally, the configuration with the smallest predicted average EPI is selected as the SUC.

The proposed *SaC* scheme is evaluated on an Odroid XU3 platform using applications

from the Rodinia [24] benchmark suite. First, two oracle schemes, *SaC optimal* and *Static optimal*, are used as references of upper bounds, where both schemes have future knowledge and employ exhaustive search in the configuration space. *SaC optimal* finds SGC and SUC pairs and the points where to switch between SGC and SUC, while the *static optimal* finds the best single configuration that meets the deadline with minimum energy. The deadline used in this evaluation is set to the fastest configuration using LITTLE cores i.e. 4-Threads, 1.4 G-Hz. The deadline is set equal to $0.7 \times$ **slowest iteration** at this configuration. This deadline restricts the usage of LITTLE cores in normal scenarios.

SaC optimal saves 10% more energy compared to *Static optimal* using race-to-idle as a baseline. This experiment shows the additional potential of energy savings that can be exploited by using slack. Another way of analyzing the behaviour of both schemes is to look at the usage of energy efficient configurations, for example those comprising of LITTLE cores. The *SaC optimal* and *Static optimal* use the LITTLE cores for 43% and 10% of iterations, respectively. This shows that *SaC optimal* is capable of exposing the most energy-efficient configurations that do not meet the deadline without exploiting slack.

Next, *SaC* is compared with *SaC optimal* and Li (a scheme presented in state-of-the-art [22]). *SaC* saves 62% and 27% more energy compared to *race-to-idle* and *Li*, respectively. Moreover, *SaC* is only 8% worse off than *SaC optimal*. The energy savings of SaC can be divided into two parts. First, a considerable amount of energy savings (i.e. 48%) come from selecting a suitable SGC. Second, further energy savings (i.e 14%) are achieved by using slack to expose the energy-efficient configurations in SU mode.

Chapter 3

Concluding Remarks

This thesis focuses on the prospect of using Quality of Service (QoS) in resource management decisions to improve energy efficiency in heterogeneous multicore systems. Specifying QoS, in terms of computational deadlines, enables resource managers to trade lower performance for higher energy savings under QoS constraints (e.g. meeting a certain frame rate). This thesis considers both single- and multi-threaded applications and a multi-dimensional configuration space where Voltage-Frequency settings (DVFS), the type of cores and the number of cores are considered.

A first contribution of the thesis (detailed in *Paper I*) is a framework for monitoring and predicting the execution time of applications at the granularity of loop iterations. By making a resource allocation decision at each loop-iteration boundary, using the proposed techniques for execution-time progress tracking and prediction across the configuration space (using DVFS and core type), the thesis shows that quite significant energy savings can be enabled.

The second contribution of the thesis (detailed in *Paper II*) is a resource management policy that not only selects an energy-efficient configuration, in the first place, but also uses the accumulated execution-time slack to expose configurations that do not meet QoS specifications. This technique considers multi-threaded applications and extends the configuration space with the number of cores allocated to them. The thesis shows that this method can save substantially more energy than a technique that optimally selects a configuration that meets the QoS demand after each iteration. Additionally, the thesis also contributes with a technique for on-line exploration of a configuration space with low overhead (detailed in *Paper II*). Low overhead is obtained by using interpolation techniques that can effectively prune the number of configurations that need to be evaluated in detail.

As for future work, first, since the technique exploiting execution-time slack builds on accurate performance and energy prediction methods, I would like to develop a more accurate, yet low overhead prediction method. It would be interesting to observe whether the increase in prediction accuracy translates into further energy savings. Second, in *Paper II*, the upper and lower limit of slack accumulation (referred to as guard-band) is fixed to an arbitrary value. However, this approach is not optimal. I would like to study how to tune the guard-band limits to application phases at run-time, instead of using a fixed guard-band. Finally, I would also like to use the insights of the papers contained in this thesis to develop resource management techniques for multiple multi-threaded programs executing simultaneously on a heterogeneous multicore platform.

Bibliography

[1] R. Dennard, F. Gaensslen, V. Rideout, E. Bassous, and A. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 10 1974.

[2] M. Bohr, "A 30 Year Retrospective on Dennard's MOSFET Scaling Paper," *IEEE Solid-State Circuits Newsletter*, vol. 12, no. 1, pp. 11–13, 2009.

[3] S. Kaxiras and M. Martonosi, *Computer Architecture Techniques for Power-Efficiency*. Morgan & Claypool Publishers, 1 2008, vol. 3, no. 1.

[4] M. Själander, M. Martonosi, and S. Kaxiras, *Power-Efficient Computer Architectures: Recent Advances*. Morgan & Claypool Publishers, 12 2014, vol. 9, no. 3.

[5] K. Olukotun, L. Hammond, and J. Laudon, *Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency*. Morgan & Claypool Publishers, 2007, vol. 2, no. 1.

[6] I. Lin, B. Jeff, and I. Rickard, "ARM platform for performance and power efficiency - Hardware and software perspectives," *2016 International Symposium on VLSI Design, Automation and Test, VLSI-DAT 2016*, pp. 1–5, 4 2016.

[7] M. Ditty, T. Architecture, J. Montrym, and C. Wittenbrink, "Nvidia's tegra k1 system-on-chip," in *2014 IEEE Hot Chips 26 Symposium (HCS)*, 2014, pp. 1–26.

[8] M. Kondo, H. Kobyashi, R. Sakamoto, M. Wada, J. Tsukamoto, M. Namiki, W. Wang, H. Amano, K. Matsunaga, M. Kudo, K. Usami, T. Komoda, and H. Nakamura, "Design and evaluation of fine-grained power-gating for embedded microprocessors," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014*, 2014, pp. 1–6.

[9] Hailin Jiang, M. Marek-sadowska, and S. Nassif, "Benefits and Costs of Power-Gating Technique Abstract," *2005 International Conference on Computer Design*, pp. 559–566, 2005.

[10] A. Raghavan, L. Emurian, L. Shao, M. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. K. Martin, "Utilizing Dark Silicon To Save Energy With Computational Sprinting," *IEEE Micro*, vol. 33, no. 5, pp. 20–28, 9 2013.

[11] B. Su, J. Gu, L. Shen, W. Huang, J. L. Greathouse, and Z. Wang, "PPEP: Online Performance, Power, and Energy Prediction Framework and DVFS Space Exploration," *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 445–457, 2014.

[12] V. Spiliopoulos, S. Kaxiras, and G. Keramidas, "Green governors: A framework for continuously adaptive DVFS," in *2011 International Green Computing Conference and Workshops, IGCC 2011*. IEEE, 7 2011, pp. 1–8.

[13] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (PIE)," *39th Annual International Symposium on Computer Architecture (ISCA)*, pp. 213–224, 6 2012.

[14] ARM, "big. LITTLE Technology : The Future of Mobile," Tech. Rep., 2013. [Online]. Available: https://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf

[15] C. J. Hughes, J. Srinivasan, and S. V. Adve, "Saving energy with architectural and frequency adaptations for multimedia applications," in *Proceedings. 34th ACM/IEEE International Symposium on Microarchitecture. MICRO-34*, 2001, pp. 250–261.

[16] F. Kluge, S. Uhrig, J. Mische, B. Satzger, and T. Ungerer, "Dynamic workload prediction for soft real-time applications," *Proceedings - 10th IEEE International Conference on Computer and Information Technology, CIT-2010*, no. Cit, pp. 1841–1848, 2010.

[17] M. Dubois, J. Suh, C.-T. Huang, and M. Dubois, "Dynamic MIPS Rate Stabilization for Complex Processors," *ACM Transactions on Architecture and Code Optimization*, vol. 12, no. 1, pp. 1–25, 4 2015.

[18] H. Chung, M. Kang, and H.-D. Cho, "Heterogeneous Multi-Processing Solution of Exynos 5 Octa with ARM big.LITTLE Technology," Tech. Rep., 2013. [Online]. Available: https://www.arm.com/files/pdf/Heterogeneous_Multi_Processing_Solution_of_Exynos_5_Octa_with_ARM_bigLITTLE_Technology.pdf

[19] M. L. Li, R. Sasanka, S. V. Adve, Y. K. Chen, and E. Debes, "The ALPBench benchmark suite for complex multimedia applications," in *Proceedings of the 2005 IEEE International Symposium on Workload Characterization, IISWC-2005*, vol. 2005, 2005, pp. 34–45.

[20] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[21] D. De Sensi, M. Torquati, and M. Danelutto, "A Reconfiguration Algorithm for Power-Aware Parallel Applications," *ACM Transactions on Architecture and Code Optimization*, vol. 13, no. 4, pp. 1–25, 2016.

[22] J. Li and J. F. Martínez, "Dynamic power-performance adaptation of parallel computation on chip multiprocessors," *The Twelfth International Symposium on High-Performance Computer Architecture*, pp. 77–87, 2006.

[23] B. Donyanavard, T. Mück, S. Sarma, and N. Dutt, "SPARTA : Runtime Task Allocation for Energy Efficient Heterogeneous Many-cores," *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis - CODES '16*, pp. 1–10, 2016.

[24] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *IEEE International Symposium on Workload Characterization (IISWC)*, 10 2009, pp. 44–54.